Sun™ Open Net Environment (Sun ONE) Software Architecture

An Open Architecture for Interoperable, Smart Web Services

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
1 (800) 786.7638
1.512.434.1511

Please
Recycle

Introduction

The Internet has had a profound impact on user expectations. Not too long ago, a computer user was a highly trained individual. Businesses trained their users to be experts in one or two specific applications. A Web user, though, is an entirely different breed. A business doesn't have the luxury of training their Web users. Web applications must be intuitive. Or perhaps it's more appropriate to say that Web applications must be invisible. A Web user simply uses the Web to read e-mail, find directions, pay bills, approve an expense request, and more. The user doesn't view these activities as executing applications. The user is simply using services that are available on the Web. The user wants to be able to access these services from a wide variety of client devices, such as desktop systems, PDAs, mobile phones, and in-car computers. Furthermore, the user wants these services to understand and act differently according to the context of the situation. Who am I? Where am I? What time is it? Am I acting as an employee or an individual?

These new user expectations are causing businesses to change the way they build application systems. Rather than building large, monolithic application systems or desktop-oriented client/server applications, businesses are starting to build applications using a service-oriented application design. Application software is being broken down into its constituent parts--into smaller, more modular application components or services. These application services make use of infrastructure software that has also been decomposed into discrete system services. All of these discrete services can be deployed across any number of physical machines that are connected to the Internet. This modular service approach gives businesses great flexibility in system design. By reassembling a few services into a new configuration, a business can create a new business service.

Services

An application service represents some type of user or business activity, such as reading e-mail, getting a stock quote, authorizing a credit purchase, and procuring materials. A system service represents system infrastructure and management functionality, such as storage, security, transactions, messaging, and fault recovery.

A service exhibits the following characteristics:

- A service provides a interface that can be called from another program.
- A service is registered and can be located through a service registry.

Service-oriented systems are not a new concept. Popular service-oriented systems include ONC RPC, DCE, COM, CORBA, RMI, and Jini™ technologies. Although all of these systems have a lot going for them, they all require special protocols for communication. A COM client must use COM protocols to speak to a COM service. A Jini client must use Jini protocols to speak to a Jini service. Unfortunately, these special protocols aren't pervasive on the Web, and firewalls routinely block communication. So now, a new type of service-oriented system is evolving: Web services.

Web Services

A Web service represents a unit of business, application, or system functionality that can be accessed over the Web. Web services are applicable to any type of Web environment, be it Internet, intranet, or extranet, with a focus on business-to-consumer, business-to-business, department-to-department, or peer-to-peer communication. A Web service consumer could be a human user accessing the service through a desktop or wireless browser; it could be an application program; or it could be another Web service.

A Web service exhibits the following characteristics:

- A Web service is accessible over the Web.
- A Web service exposes an XML interface.
- A Web service is registered and can be located through a Web service registry.
- Web services communicate using XML messages over standard Web protocols.
- Web services support loosely-coupled connections between systems.

Perhaps what is most intriguing about Web services is that it doesn't matter what technologies are used to build them. Because Web services communicate over standard Web protocols using XML interfaces and XML messages, all Web services environments can interoperate — at least in theory.

What Businesses Want

Since the advent of the Web in 1994, businesses have been using the Internet to get closer to their stakeholders: shareholders, employees, customers, and partners. They have been using the Internet to automate supply chains and improve business process efficiency. Electronic business is no longer a future direction; it has become the norm. Now businesses are intrigued by the Web services approach to Internet computing. The Web services computing model promises even better cross-business integration, improved efficiency, and closer customer relationships.

But in order to make the Web services model palatable, businesses must be able to leverage existing application assets and expose them as Web services. Businesses want tools and technologies that will reduce the cost, risk, and complexity of moving to this new model.

Web Services Technologies

A variety of technologies that support Web services is starting to appear. The leading candidates include UDDI, WSDL, SOAP, and ebXML. Currently, these technologies are not available as supported products, but they are maturing quickly.

- The UDDI (Universal Description Discovery and Interoperability) initiative is an industry consortium lead by Accenture, Ariba, Commerce One, Compaq, Edifecs, Fujitsu, HP, I2, IBM, Intel, Microsoft, Oracle, SAP, Sun Microsystems, and VeriSign. More than 130 companies have joined the UDDI initiative. The group is developing specifications for a universal, Web-based business directory called the UDDI Business Registry. At the time of this writing, the UDDI V1 specification is available, and the UDDI Business Registry is undergoing beta testing. Ariba, IBM, and Microsoft jointly operate the UDDI Business Registry. IBM has released an open source UDDI for Java™ technology developer toolkit through the IBM developerWorks Open Source Zone.
- WSDL (Web Services Description Language) is technology developed by Ariba, IBM, and Microsoft. It specifies a common XML framework for describing a Web service. At this time, IBM has released a WSDL for Java technology developer toolkit through IBM alphaWorks.

- SOAP (Simple Object Access Protocol) is technology developed by DevelopMentor, IBM, Lotus, Microsoft, and Userland. SOAP provides an extensible XML messaging protocol and also supports an RPC programming model. A number of SOAP implementations are available. The two most popular implementations are an open source Java technology implementation from the Apache Software Foundation and a Microsoft implementation within the .NET SDK. These two implementations are fairly stable, although interoperability problems between the implementations continue to plague developers.
- An extended variant of SOAP, called SOAP Messages with Attachments, has been developed by HP and Microsoft. Microsoft BizTalk Server 2000 uses SOAP Messages with Attachments, so this technology is included in a supported product. The W3C (World Wide Web Consortium) recently formed an XML Protocol Working Group to develop a standard XML messaging protocol (XP). The SOAP developers submitted the SOAP specification to the W3C, who are using it as a starting point for the XP effort. SOAP Messages with Attachments has also been submitted to the W3C. Details of W3C work in progress are not available to the public.
- ebXML (Electronic Business XML) is a B2B XML framework being developed by the ebXML Initiative. The ebXML Initiative is a joint project of UN/CEFACT (the United Nations body for Trade Facilitation and Electronic Business) and OASIS (Organization for the Advancement of Structured Information Standards). The ebXML membership includes representatives from more than 2000 businesses, governments, institutions, standards bodies, and individuals from around the world.
- ebXML is a complete B2B framework that enables business collaboration through the sharing of Web-based business services. The framework supports the definition and execution of B2B business processes expressed as choreographed sequences of business service exchanges. The framework includes specifications for a Message Service, Collaborative Partner Agreements, Core Components, Business Process Methodology, and Registry and Repository. A proof-of-concept interoperability demonstration was presented in October 2000. Participants included Ajuba Solutions, Cisco, Extol, Fujitsu, IBM, IPNet, Netfish, NTT, Savvion, Sterling Commerce, Sun Microsystems, TIE, Viquity, WebMethods, XML Global, and XML Solutions. The ebXML specifications are scheduled to be published by May 2001.

The Developer's Dilemma

Although these technologies are rapidly maturing, nascent Web services developers are pretty much left to their own devices to figure out how to make Web services work. There are no guidelines to help put all of these technologies and standards into perspective. They can use a context-sensitive XML editor to manually define a Web service interface, or generate one using a SOAP or WSDL toolkit. But how do developers associate a new Web service interface with an existing application or service? And how do they assemble multiple discrete Web services into a composite business service? Interoperability between vendor implementations is still challenging if not impossible. The vision of seamless assembly and integration of distributed, heterogeneous Web services is still a distant dream.

Shared Context

The issue is a matter of shared context. Context refers to the things that a Web service needs to know about the service consumer to provide a customized, personalized experience. Context refers to such things as the identity of the consumer, the location of the consumer, and any privacy constraints associated with the consumer information. If a number of services are aggregated to create a composite business service, these services somehow need to share this context.

Take for example shared user identity. A Web service that offers a personalized experience maintains information about the identity of each user. When an individual uses that service, she identifies herself to that service. Sometimes the service determines her identity automatically from a cookie maintained on her system. Sometimes she has to enter a user ID and password. Consider for a moment how many user IDs and passwords you have to remember for all of the Web sites you visit. Consider how many more cookies you have stored on your system. Consider how many businesses maintain their own database of private information about you.

This discussion has no doubt raised your hackles about privacy and security issues. It also points out a significant challenge facing Web services technology. There are no standards that allow Web sites to share something as simple as user identity. There are, in fact, no standards to represent user identity. There are no standards that ensure user privacy. How can the seamless integration of Web services occur if these services can't safely share identity as well as other situational context?

Before the vision of transparent, dynamic interaction of widely distributed, heterogeneous Web services can be achieved, this issue of shared context must be solved. Shared context raises some fairly sensitive issues. The solution can't come from a single vendor. It cannot be proprietary. The solution must be open and interoperable. It must work with any Web service. And the solution must ensure the security and privacy of individual information.

What Developers Want

So where does this leave the developer? Obviously developers aren't going to sit back and wait for new standards to be developed. They need to start building Web services today. But they want help. They want to understand how all the different Web services technologies work together. They want guidelines to help ensure that Web services are interoperable. And they want to be able to build "smart" Web services.

Smart Web Services

A smart Web service is a Web service that can understand situational context and share that context with other services. It produces dynamic results based on who, what, when, where, and why it was called. A smart Web service can be aware of a number of situational circumstances, such as:

- The service consumer's identity, whether it be an individual, a business, or another Web service
- The role that the consumer is using at the time it invokes the service
- Preferences that the consumer may have defined for this type of service
- The security policies associated with the consumer of this service
- The privacy policies associated with the consumer
- The business policies associated with the consumer of this service
- The physical location of the consumer
- The type of client device being used to invoke the service
- Any past history associated with the consumer of this service or related services
- Any service level agreements that exist between the consumer and this service provider

For example, let's say you invoke a smart restaurant finder service to help you select a restaurant. The smart restaurant finder wants to make a recommendation based on where you are, what you like, where you ate last, and whom you're with. It makes a big difference if you're going out with your children or with an important prospective client. So the smart restaurant finder considers your identity, your preferences, your history, and your role.

As another example, let's imagine that there's a power emergency in California. The electric company broadcasts a Stage III emergency electrical supply shortage alert to all power consumers. This notification would launch an energy conservation service, if available, at all consumer sites. In a business facility, the energy conservation service would automatically reduce power utilization by adjusting thermostats and switching off lights according to predefined policies based on parameters, such as time of day and day of the week. Some of these predefined policies might be superseded by situational parameters such as a temporary service level agreement that's in effect during a particular production run. In a home, the energy conservation service could adjust the thermostat, switch off lights, and impose a volume governor on your teenager's stereo. But the smart Web service also considers who is physically present in the home. For example, the smart service knows that the baby is sleeping upstairs, and therefore doesn't adjust the thermostat for the upstairs heating zone.

As a final example, let's consider an order fulfillment process. Imagine that a customer has ordered six different items. At the time of the order, you promised the customer that the goods would be shipped within two days. But as your procurement service places the order with your contract suppliers, it determines that there is a supply constraint on one of the items that will prevent you from shipping the entire order as promised. Your standard policy in a situation like this is to send partial shipments, but this policy eats away at your profit line. What's most irritating is that even if you send multiple shipments, many times they all arrive on the same day, and the customer may not care if the goods arrive today or tomorrow. You're looking for an alternative that saves you money and yet still keeps the customer happy.

Rather than automatically sending the goods in multiple shipments, a smart Web service would determine a course of action by considering the customer profile and past history, and by weighing the profit impact of expediting the missing components or reshuffling factory work orders. The service may also take into consideration other parameters such as cost of inventory, cost of space, estimated shipping charges, and expected delivery times. The smart Web service might send an inquiry to the customer by e-mail, SMS message, automated voice response message, return fax, or some other delivery mechanism as defined in the customer profile--requesting a decision:

1. Please accept this coupon for your inconvenience; the goods will be shipped when the order is complete, which is currently estimated to take five days.

2. Ship the order in pieces as they become available.

Making Web Services Smarter

Anyone can build context-sensitive Web services. Certainly there are many Web sites today that offer highly customized content based on who you are and your past history with the site. But context-sensitivity is only one half of the problem. A smart service must also be able to share context with other services. Unfortunately, there are no standards or conventions for representing this context. Every site that provides a personalized experience maintains identity and history information in a proprietary format.

What's needed is a new set of standards, an XML framework, to represent contextual information. What's also needed is an open architecture that defines how services use this information and assures service interoperability.

An Open Architecture for Interoperable, Smart Web Services

Sun has defined an open software architecture to support interoperable, smart Web services. The Sun™ Open Net Environment (Sun ONE) software architecture addresses important issues such as privacy, security, and identity. It defines practices and conventions to support situational context, such as client device type and user location. And it supports systems that can span multiple networks, including the traditional Web, the wireless Web, and the home network. The architecture is designed to ensure that smart Web services, developed using any tool, running on any platform, can seamlessly interoperate.

The Sun ONE architecture is based on a recommended set of open and pervasive standards, technologies, and protocols. While defining the architecture, Sun identified some new standards that are needed to complete the architecture. Sun will collaborate with other companies to foster the development of these new standards through appropriate standards bodies and industry initiatives, such as W3C, OASIS, IETF, UDDI, ebXML, and the Java Community Process™ Program. In particular, Sun will work with others to foster the development of new standards to support shared context.

Core Standards and Technologies

At its essence, the Sun ONE software architecture is based on XML, Java technology, and LDAP. Sun's technical philosophy is to use common, pervasive technologies, to use what is available, and to not reinvent the wheel. The core standards used within the architecture are:

- XML standards and initiatives, including the core W3C XML recommendations (XML, DOM, XML Namespaces, XSLT, XPath, XLink, and XPointer); the development community XML parser, SAX; presentation formats (XHTML, WML, and VoiceXML); XML schema systems (DTD, XML Schema, RELAX, and the emerging TREX specification); XML messaging systems (the emerging W3C XML Protocol, SOAP, and ebXML Message Service); XML registries and repositories (UDDI Business Registry, ebXML Registry and Repository, and OASIS xml.org); XML service description languages (WSDL); XML management information interchange frameworks (DMTF CIM and WBEM); B2B XML frameworks (ebXML Collaborative Partner Agreement, ebXML Business Process Methodology, and ebXML Core Components); and XML security systems (XML Signature, XML Encryption, and the emerging OASIS security services effort).
- Java technologies, including the Java platforms (Java 2 Platform Enterprise Edition (J2EE™), Java 2 Platform Standard Edition (J2SE™) Java 2 Platform Micro Edition (J2ME™), the Mobile Information Device Profile, and Java Card™ API), Java Servlet and JavaServer Pages™ (JSP™) software, the Java API for database access (JDBC), and the Java APIs for XML (Java API for XML Processing, Java API for XML Data Binding, Java API for XML Registries, and Java API for XML Messaging).
- Infrastructure standards, such as HTTP, SLL, and LDAP.

Sun ONE Software Architecture

The Sun ONE software architecture is a living system, and it will continue to grow as new technologies appear that might enhance the environment. The illustration in Figure 1 presents a high-level functional overview of the major components of a Web services architecture.

Across the top of the illustration is a box entitled Service Creation and Assembly. This part of the architecture represents the tools used to develop systems based on the Web services model. A Web service is often made up of a number of discrete service components that instantiate various bits of content, business logic, and applications. Therefore, the service development process involves two distinct steps.

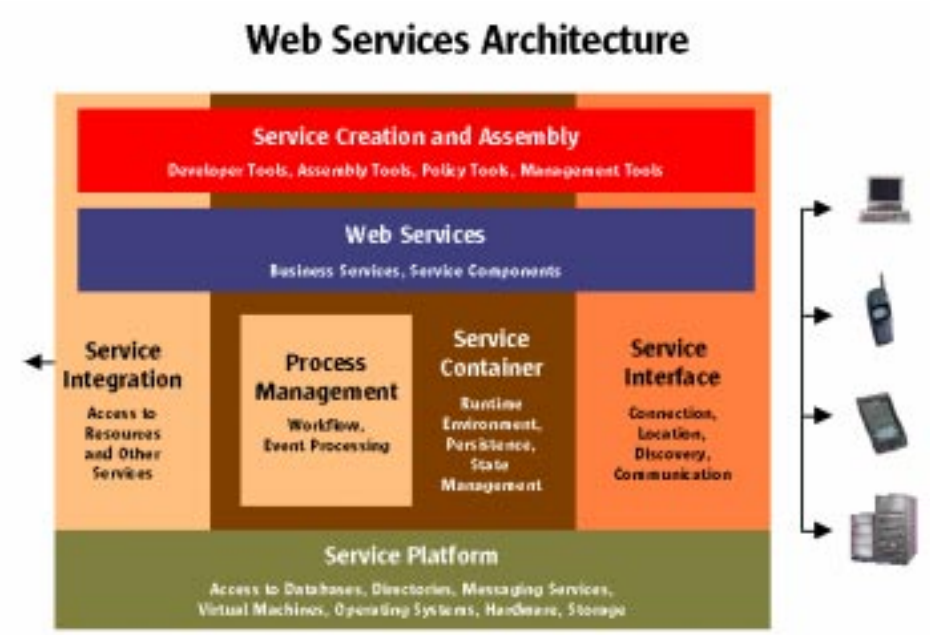

Figure 1: A functional overview of the major components of a Web services architecture.

The first step involves creating the discrete services, which the Sun ONE software architecture refers to as micro services. The second step involves assembling the micro services into composite services, or macro services. Developers create micro services using integrated development environments, code generators, XML editors, and authoring tools. (More information about the micro service development model appears later in this paper.) Service assemblers use business process modeling tools, workflow tools, and other types of glue tools to assemble macro services. Assemblers also use policy tools to define business rules, security policies, and context-sensitive policies that can dynamically change the macro service process at runtime. Once the services are tested and complete, the services are ready to be deployed to the deployment platform, and the policies and rules can be deployed to an open directory. Management tools provide facilities to deploy and manage the service environment.

The box under the service creation and assembly box in the illustration in Figure 1 represents the deployed services, consisting of business services (macro services) and service components (micro services). Micro services can be assembled, configured, and reconfigured into any number of macro services.

The remainder of the illustration in Figure 1 represents the Sun ONE software deployment architecture. Working from right to left, the diagram depicts service consumers interacting with a service interface. The service interface provides basic connection, location, discovery, and communication functionality. This part of the architecture includes all types of user and business portals that enable Web users, wireless users, voice users, and external business systems to invoke services. The service interface directs each request to the appropriate Web service.

Macro services and micro services execute within some type of service container. A service container provides a runtime environment for the service and provides persistence and state management services for the service. The service container runs on a service platform, which provides access to databases, directories, and messaging services. The service platform resides on an operating system or virtual machine, providing access to hardware, storage, and networks. Note that a Web service can be deployed on any type of platform or intelligent device.

The process management facilities work in concert with the service container to manage service workflow and event processing. The service integration facilities use the underlying service platform to access other Web services and resources such as databases, files, directories, and legacy applications.

Adding Smarts to the System

The illustration in Figure 2 shows an enhanced Web services architecture, which includes additional functionality to support context sensitivity. Smart policy coordinates activities according to policies associated with identity, context, and roles. Smart delivery can aggregate, customize, and personalize service results based on context. Smart process uses context to affect business service workflow. Smart management ensures privacy, security, and access rights based on the specific situation as defined by the context.

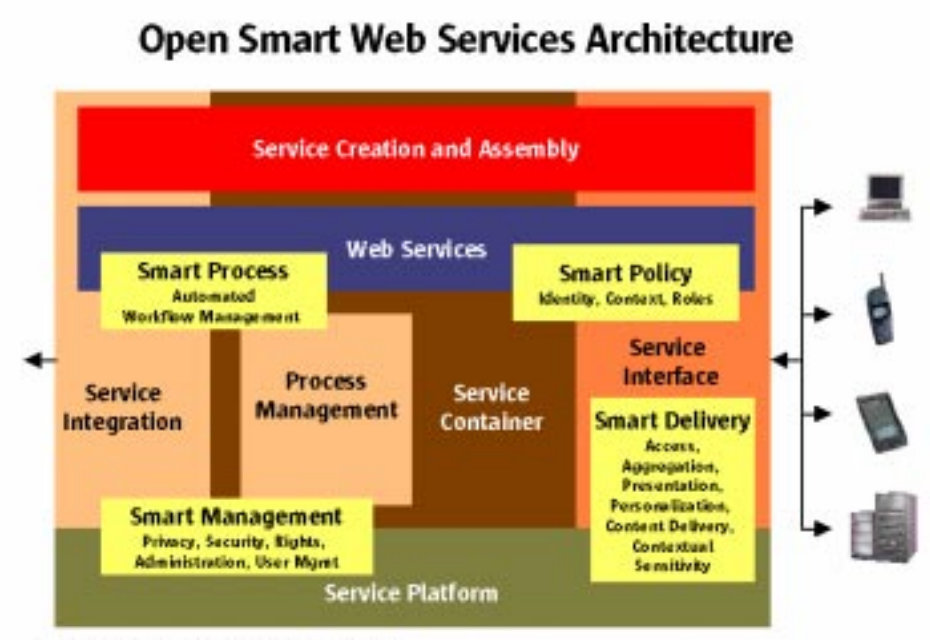

Figure 2: Making Web services smarter.

Web Services Processing Model

Figure 3 shows a conceptual overview of the processing model based on the smart Web services architecture.

Preparing a Service

The smart management facilities manage, monitor, and maintain a Web service. These facilities

- Ensure that the service is properly registered and locatable through one or more service registries
- Ensure that appropriate pay-per-use or subscription agreements are in place and properly executed
- Coordinate the provisioning of a service, and ensure that the service performs according to a minimum quality of service as determined by service level agreements or other criteria

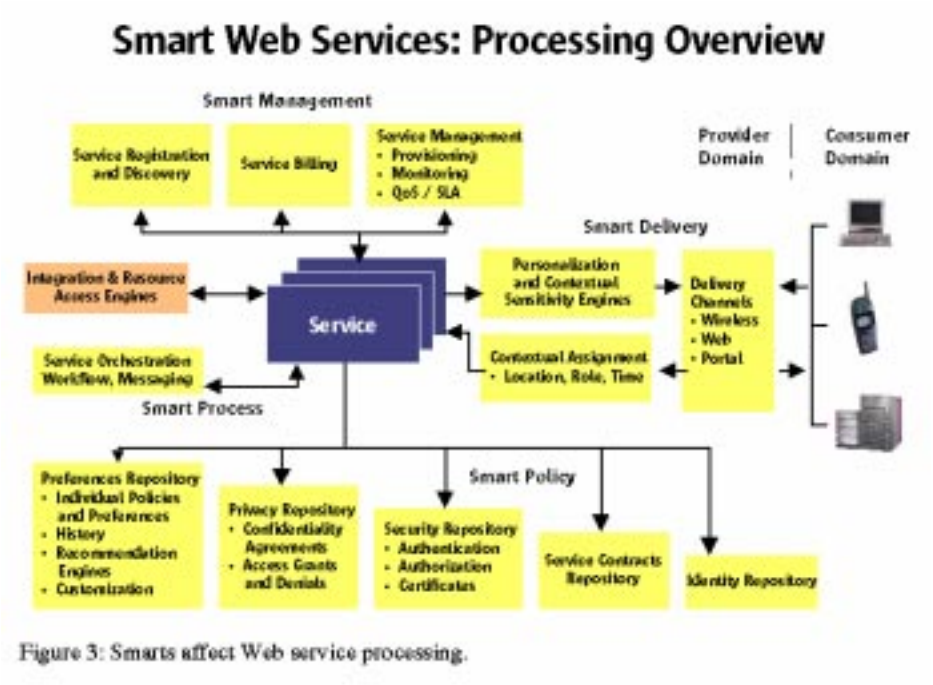

Figure 3: Smarts affect Web service processing.

- Obtain management and runtime policies from the smart policy facilities.

The smart policy facilities. manage and interface with the various directories and repositories that contain the policies and business rules that affect service processing.

Processing a Service Request

A service request can come in through any type of delivery channel. The smart delivery facilities determine the current context of the request, record information such as location, role, and time, and pass the request to the service.

Service Workflow

The smart process facilities manage the choreography of a macro service. Based on the contextual assignments and the policies and business rules defined for the service, the smart process facilities ensure that the appropriate micro services or external services are invoked in the proper sequence. The services use the integration and resource access engines to invoke other services and to access databases, files, legacy applications, and other resources.

Delivering a Service Response

When processing is complete, the smart delivery facilities tailor the response for the consumer using personalization and contextual sensitivity engines. The final response is delivered back to the user through the appropriate delivery channel.

Standards Backplane

The illustration in Figure 4 shows the Web services architecture overlaid with many of the standards and technologies that provide the foundation for the Sun ONE software architecture. Not all of these standards are required in every case. The standards are provided as a guideline and an aid for interoperability.

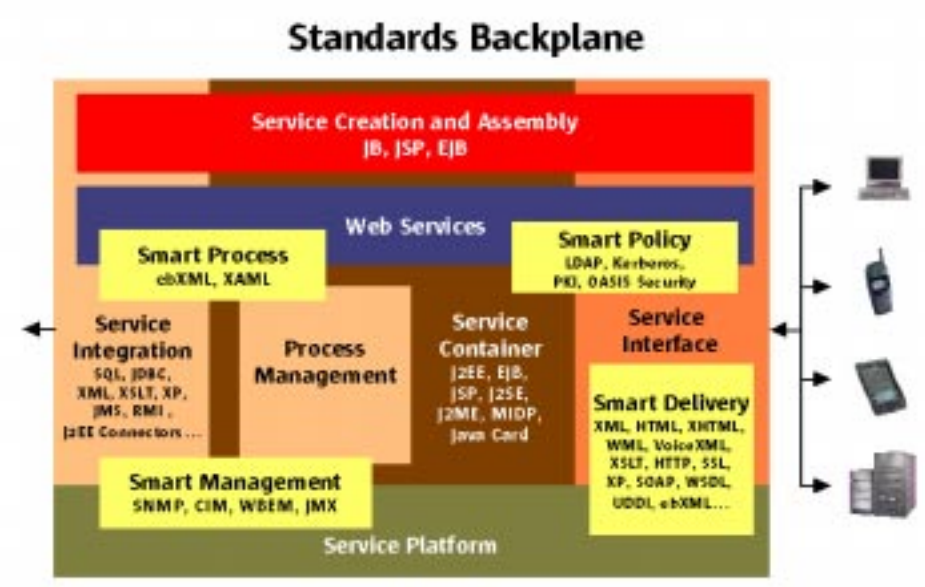

Figure 4: Standards ensure interoperability.

Smart Delivery

Smart delivery supports a variety of clients using a number of device-specific presentation formats, including HTML, XHTML, WML, and VoiceXML.

- The Hypertext Markup Language (HTML) is the most commonly used technology for presenting electronic information to users.

- XHTML is an XML-compliant variant of HTML. HTML and XHTML are efforts of the W3C.
- The Wireless Markup Language (WML) is the presentation format used on many WAP-compliant wireless devices such as mobile phones. WML is an effort of the WAP Forum, which has recently adopted a subset of XHTML as a follow-on to WML.
- VoiceXML allows for the creation of audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed-initiative conversations. Its major goal is to bring the advantages of Web-based development and content delivery to interactive voice response applications. VoiceXML is a joint effort of the VoiceXML Forum and the W3C.

Smart delivery also manages content transformation, generally using XML and XSLT. The Extensible Markup Language (XML) is a platform- and language-neutral data representation format, and it forms the underpinning for every important effort focused on Web services and electronic business. Applications manipulate XML using either DOM or SAX. The Document Object Model (DOM) is a platform- and language-neutral API that allows programs to dynamically access and update the content, structure, and style of XML and HTML documents. DOM provides in-memory, tree-structured access to XML data so that the entire structure of a document is accessible for reading and writing. The Simple API for XML (SAX) provides an event-based serial XML parser. Extensible Stylesheet Language Transformations (XSLT) is a language for programming the transformation of XML documents to a variety of formats, such as XHTML, WML, VoiceXML, other XML documents, or to other data formats. XML, DOM, and XSLT are work efforts of the W3C. SAX is a collaborative effort by the members of the XML-DEV discussion group.

A Web service exposes an XML interface, which is implemented using an XML messaging protocol such as SOAP, ebXML Message Service (ebXML MS), or the emerging W3C XP. An XML schema language, such as Document Type Definition (DTD), XML Schema, RELAX, or the emerging TREX specification, describes the format of the XML message. The message schema should be registered in a schema registry such as xml.org. The Web service can be described using a Web service description language, such as WSDL, and the service can be registered in the UDDI Business Registry and/or in the ebXML Registry and Repository.

Service Container

The service container provides the runtime environment for Web services. The type of service container used depends on the type of platform being used to host the services. The Sun ONE software architecture provides recommendations for both servers and devices.

The recommended service container on a server is a J2EE application server. J2EE technology-based services can be implemented as servlets, JSP pages, or Enterprise JavaBeans™ (EJB™) components. Currently, J2EE application servers transparently manage much of the deployment and execution complexity associated with Web services. As the Web provides more context information for requests as well as information about the services that are available, the J2EE container will play a fundamental role in transparently using this information to configure and customize Web services. Instead of requiring complex programming by the developer, the J2EE container will extend its existing declarative approach to automate the development and deployment of smart Web services. This declarative approach has already been applied to the areas of resource management, transaction management, state management, and security. J2EE declarative programming significantly reduces the complexity of Web service development.

The recommended service container on a device is J2ME technology, which provides a deployment platform for Web services on consumer devices such as wireless phones, PDAs, televisions, automotive information systems, and a wide variety of other consumer and embedded devices. A J2ME device is defined in a profile, which includes a Java virtual machine and a collection of APIs that provide access to the capabilities of the underlying device. Profiles provide the necessary abstractions allowing transparent deployment of services and content across an extremely wide variety of networks and device types. Technologies currently available include the Mobile Information Device Profile (MIDP) for wireless phones and PDAs, and the Java TV™ API for broadcast televisions, set-tops, and personal video recorders. These platforms provide application models that enable services to access the unique features of the device, including storage, user interface, and personalization services.

Smart Management

Traditionally, system and network management has been implemented using a large, distributed application, where special agent software from a system management software vendor is installed on each resource to be managed, and a central server made by the same company connects to each agent and manages the resource. Although most management agents conform to the Simple Network Management Protocol (SNMP) standard, most management systems encourage a homogeneous management infrastructure to reap the most benefit. This approach has led to large, expensive-to-deploy system management frameworks that lock customers into one vendor for all aspects of management.

Today, system management is undergoing a transition to Web services based on industry-wide standards, which will enable a new generation of system management solutions that are modular and interoperable. In this new model the proprietary agents are replaced by agents designed to use open standards such as XML and HTTP, and the central server is replaced by Web services that use these standards to interact with the agents.

In this model, system and network equipment will provide management agents for their devices, and a wide variety of companies will compete with modular, Web services targeting specific aspects of management. So instead of customers being locked into a single vendor, they will be able to go to best-of-breed vendors for each aspect of management.

The standard for this model is called the Common Information Model (CIM). Defined in the Distributed Management Task Force (DMTF), a standards body focused on distributed management, CIM provides a similar role for system management that ebXML does for business to business transactions. CIM defines the schema and protocol standards that enable the design of Web services that are able to work independently of the underlying design of the other services and agents with which they interact. For example, CIM defines the standard XML objects for accessing basic server information over a network, and many of these objects are already implemented in volume server operating platforms such as the Solaris™ Operating Environment and Windows 2000.

This model brings about two important aspects for the world of open, smart Web services:

1. Web services will be managed by this model. Developers can instrument a Web service using the Java Management extensions (JMX™) API and create standard interfaces for managing the deployment, health, and performance of the service. The CIM standard enables this work to be done by the people who are in the best position to optimally implement these features — the developers. As Web services are deployed, Web-based management services will be able to use these same standards to discover these services and manage them through their agents.

2. .As a well-defined service, companies will be able to outsource aspects of their system and network management to third-party management service providers (MSPs) who can perform those same tasks over the network. Since these services are modular, customers can determine on a case-by-case basis which parts of management they want to retain in-house, and which parts they would like to outsource.

Smart Process

The smart process facilities enable context-sensitive service choreography. Context-based smart processing can change the outcome of a macro Web service by dynamically altering the choreographed sequences of micro Web service invocations based on the context of the request, such as geography, jurisdiction, or the maturity of the business relationship.

In an ebXML environment, a B2B business process is expressed as a choreographed sequence of business service exchanges. Each basic service exchange is referred to as a business transaction. A business service is typically a high-level service, such as an ordering service or a billing service. But behind the scenes, that one business service is recursively composed of a number of lower level services, such as a product look-up service, a pricing service, and a currency conversion service.

The Transaction Authority Markup Language (XAML) provides an alternate method to choreograph business services. The XAML initiative is sponsored by Bowstreet, HP, IBM, Oracle, and Sun Microsystems. XAML defines a set of XML message formats and interaction models, and enables the coordination and processing of business-level transactions that span multiple parties across the Internet.

Smart Policy

A Web service can use a policy engine to dynamically adapt processing and/or results according to rules that consider user identity, authorization levels, and other contextual information. The context-oriented standards have yet to be defined, but a number of standards and proposals exist in relation to security. User and policy information is maintained in an open directory accessed using LDAP. Kerberos and Public Key Infrastructure (PKI) provide services for authentication, authorization, and encryption. In addition, the OASIS Security Services Technical Committee is working to define an XML framework for exchanging authentication and authorization information so that cross-enterprise transactions can be made more secure using open technology. This framework will provide a request/response protocol for accessing authentication and authorization services, encode requests for and assertions of identity and entitlement in XML form, and supply bindings of the security messages to various transport and messaging protocols. The OASIS committee is combining two proposed security specifications: the Security Services Markup Language (S2ML) and AuthXML. The S2ML specification was written by Bowstreet, Commerce One, Jamcracker, Netegrity, Sun Microsystems, VeriSign, and WebMethods. The AuthXML specification was written by an industry consortium lead by Outlook Technologies.

Web Services Developer Model

Although Web services can be developed in any programming language, the Sun ONE software architecture is based on the Java platform. The Java platform includes native support for XML. The Java API for XML Processing (JAXP) provides a Java interface to DOM, SAX, and XSLT. Additional Java APIs for XML are in various stages of development through the Java Community Process Program. The Java API for XML Data Binding (JAXB) binds XML data to Java code. A developer uses JAXB to compile XML schema information into Java objects. At runtime, JAXB automatically maps the XML document data to the Java object and vice versa. The Java API for XML Messaging (JAXM) provides a native Java interface to XML messaging systems such as ebXML MS, W3C XP, and SOAP. The Java API for XML Registries (JAXR) provides an interface to XML registries and repositories such as the ebXML registry/repository and the UDDI Business Registry. A new Java Specification Request has just been submitted to the JCP to define the Java APIs for XML-based RPC (JAX/RPC), which would provide direct support for an RPC programming convention for XML messaging systems, such as SOAP and the forthcoming W3C XP.

The Anatomy of a Web Service

Figure 5 shows a micro Web service from the developer's perspective. A Web service consists of a Web service interface and one or more service components. The Web service interface manages and manipulates XML messages. The service components contain the business logic that implements the service. The service components frequently interact with external resources and services through a variety of integration services.

Web Services Interface

A Web service communicates by passing XML documents over standard Web protocols, such as HTTP. The Web service interface implements the code that produces and consumes these XML messages. The Sun ONE software architecture recommends three types of XML messaging systems: SOAP, ebXML MS, and the future W3C XP.

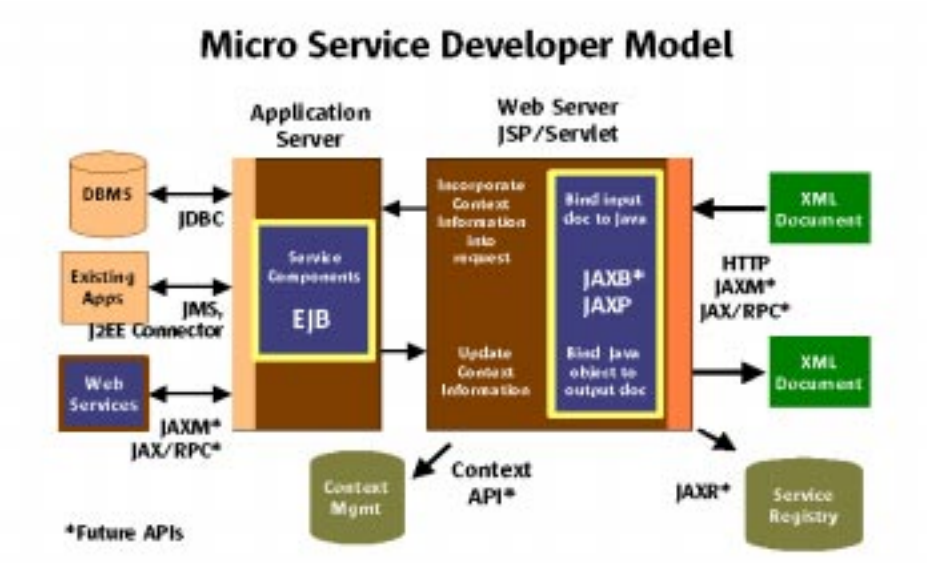

Figure 5: Current and future Java APIs.

- SOAP is a lightweight, extensible XML messaging protocol. Although SOAP is a general-purpose XML messaging services, it is particularly attractive for RPC-style service invocations. A SOAP message is an XML document consisting of a header and a body. The message data are specified in XML and packaged in the SOAP body. SOAP supports an RPC programming convention that automatically binds input and output parameters to XML elements in the SOAP message body. Since an XML document cannot contain another XML document, a SOAP message cannot contain a complete XML document, such as a purchase order. The SOAP protocol also does not support attachments of multimedia files or other non-XML data. SOAP does not provide a quality of service (QoS) framework. Java tools for implementing SOAP interfaces are available from the Apache Software Foundation. The basic SOAP environment can be extended to support a QoS framework, but no implementations are available at this time.
- ebXML MS is an XML messaging service designed to support the requirements of B2B ecommerce. An ebXML message is a multipart/MIME-encapsulated message that can transport any number of XML documents and non-XML attachments. ebXML MS does not support an RPC programming convention. ebXML MS provides a QoS framework that ensures reliable message delivery. The QoS framework can be extended to support security and transaction semantics. The ebXML specifications will be available in May 2000.
- The W3C XP specification is under development. Details of W3C work in progress are not available to the public.

A Web service interface is implemented as a JSP page or servlet running within a Web server. The JSP page/servlet receives the XML message and extracts the XML document. Currently, this process is performed manually. In the future, developers will be able to use JAXM or JAX/RPC to more efficiently process the XML message. The JSP page/servlet then takes the XML document and maps the document data into Java object data. Today, developers can use JAXP to process the XML document. In the future, the developer will be able to use JAXB to automatically bind the document to a Java object.

Capturing Context Information

The Web service interface also captures as much context information as is available. User identity can be captured by retrieving a cookie from the client's system or challenging the user for a userid and password. Once the service interface has determined identity, the service can query its proprietary user history files to retrieve additional context. Sun is collaborating with partners to foster a standards effort to define standard frameworks to deal with identity and context. The details of the context framework have yet to be defined, but it is expected that there will be standard APIs that the JSP page/servlet would use to obtain user context.

Business Logic

Once the JSP page/servlet has mapped the XML data to Java data and captured the context, it then performs its usual session-management services and invokes the business logic. The business logic performs the actual service. The business logic is implemented as an EJB component, or possibly a servlet for light-duty applications. The business logic component will interface with a workflow engine that manages the choreography of the composite business transaction, and with a policy engine that applies appropriate business rules according to user identity and context. Standards for this process have yet to be defined. No doubt, ebXML, XAML, and the OASIS security services will play a role in this process.

Integration Services

The business logic component will need to interface with various resources. The JDBC™ APIprovides access to databases; J2EE Connectors and Java message Service (JMS) provide access to other application systems; and XML messages are used to interface with other Web services. For the moment, developers will need to construct XML messages manually using JAXP. In the future, developers will use JAXM or JAX/RPC to more easily manipulate XML messages.

Returning Results

When the business logic component has completed its work, it returns a Java result object to the JSP page/servlet interface component. The interface maps the results from the Java object to the output XML document using JAXP or, in the future, JAXB. The JSP servlet can then use the context to further customize the response document as appropriate. When ready, the response is returned to the requester.

Summary

The Sun ONE software architecture is designed to help developers be successful building Web services today. The Web services technologies that are available today are still rudimentary, but that fact shouldn't stop developers form venturing into this exciting new territory. Web services represent the next generation of software. This architecture provides a guideline to help developers put the myriad XML standards, technologies, and initiatives in perspective. The developer model provides a foundation for development efforts, indicating which technologies and APIs should be used for each facet of a Web service. Sun will continue to provide tools, technologies, specifications, and advice to promote the Web services model of computing. .

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

1 (800) 786.7638
1.512.434.1511

http://www.sun.com/software/sunone/

February 2001